Práctica 4:

ALGORITMO MERGESORT

OBJETIVO

- Estudiar un ejemplo de esquema algorítmico "Divide y Vencerás".
- Estudio de un algoritmo de ordenación rápida.
- Evaluación del coste de funciones recursivas.

MATERIAL A ENTREGAR

- merge.cpp
- Gráficas (merge_t.png, merge_r.png)
- makefile
- Ejercicios apartado 1. **(previo)**

DESARROLLO

1. Ejercicios previos

a) Adaptar el algoritmo Mergesort visto en teoría (Tema 3, apartado 4, pp 47-48) a C++.

b) Calcular el coste exacto en accesos al vector (asignaciones más comparaciones) del algoritmo Mergesort obtenido en el apartado *a* para el **caso mejor y peor**. Las recurrencias deben resolverse totalmente mediante el método de la ecuación característica.

2. Representación del coste en pasos

Implementar el algoritmo Mergesort para ordenar vectores (STL) de *VElement*. El vector auxiliar para la realización de la mezcla se deberá pasar por referencia para evitar crearlo en cada llamada recursiva.

Para comparar el análisis anterior con el comportamiento del algoritmo, se deberán utilizar los contadores adecuados de la clase *VElement* y la función **evaluar** de la práctica 2 generalizada convenientemente. Se consideran accesos tanto las comparaciones como las asignaciones, por tanto habrá que sumar los dos contadores. Habrá que obtener una tabla de talla–coste (ficheros .dat) para el **caso medio** para lo cual habrá que ejecutar un número determinado de ordenaciones aleatorias y calcular el promedio para cada talla. Se deberá realizar la tabla para tallas del problema (el tamaño del vector) de 50 a 5.000 en incrementos de 50. Esta curva empírica deberá representarse en gnuplot conjuntamente con las curvas teóricas (caso mejor y caso peor) obtenidas en el apartado 1 (merge_t.png).

3. Cálculo del coste real en segundos

Calcular el coste real en segundos del algoritmo mergesort para el caso medio. Para ello se deberá implementar una función capaz de calcular el coste en segundos de otras funciones.

Implementar la función **calculoCoste**, que permitirá volcar en un stream de salida (file) los valores (pares n t(n)) del coste en segundos de una función (f) para un rango suficiente del tamaño del problema **n** (entre ini y fin) con un incremento determinado (inc). El parámetro f será un puntero a función, lo que nos permitirá pasarle cualquier función que cumpla la especificación dada. La cabecera de esta función será:

```
void calculoCoste(unsigned ini, unsigned fin, int inc, float (*f)(unsigned),
ostream & file);
```

Para el cálculo del coste temporal de las funciones se usará la función clock_gettime() de la librería time.h. Esta función es específica de Linux y sirve para leer el tiempo de los diversos relojes definidos en el sistema. En nuestro caso lo utilizaremos para leer el tiempo de CPU del proceso desde el que se llama a la función.

Un ejemplo de cómo utilizarlo para calcular el coste temporal en segundos de una función f() sería el siguiente:

```
struct timespec ini_time, fin_time;
double time;


clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &ini_time);
for(unsigned i = 1; i <= repeticiones; i++)
    f();


clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &fin_time);
time=(fin_time.tv_sec - ini_time.tv_sec) +
     (fin_time.tv_nsec * 1.0e-9 - ini_time.tv_nsec * 1.0e-9);
cout << "Coste en segundos: " << time / repeticiones << endl;
```

Representar el resultado en la gráfica *merge_r.png*.